Formal Verification of Service-Oriented Adaptive Driver Assistance Systems

Christian Schwarz and Dieter Zöbel
Universität Koblenz-Landau, Koblenz, Germany
Email: (chrschwarz|zoebel)@uni-koblenz.de

Marco Wagner
Hochschule Heilbronn, Germany
Email: Marco.Wagner@hs-heilbronn.de

***Abstract*—Many future Driver-Assistance-Systems (DAS) will use components not permanently mounted to the vehicle. Unlike state-of-the-art DAS with static configurations, the system and software architecture changes at runtime. To handle configuration changes, Service Oriented Architecture (SOA) and automatic orchestration is a promising approach. Whenever systems are set up automatically, they have to be validated. This paper presents an approach based on formal methods. Existing component models are annotated with Quality-of-Service parameters and transformed automatically to Hybrid Automata. These automata are then composed to an overall system model and model checking is used to check safety properties. The complete transformation-orchestration-validation process is executed without user interaction and thus can be performed at runtime.**

I. Introduction

State-of-the-art Driver-Assistance-Systems (DAS) are characterized by a static system architecture. The set of electronic control units (ECUs) and the functional and electrical topology of the networking rarely changes during their life cycle. However, future DAS, like systems for truck-trailer-combinations or systems involving car-to-X communication, will base on hardware configurations and topologies that are subject to change while the system is in use. A DAS for truck-trailer-combinations for example consist of components mounted on separable parts of the vehicle. Combinations of these parts are very likely to be disconnected and reassembled in everyday life. Therefore, the software and system architecture underlying this kind of DAS needs to be able to handle changes at runtime. In [1] we presented an approach capable of handling such changes using Service Oriented Architecture (SOA).

The paradigm of Service Orientation is a commonly used approach within the field of distributed software systems. It uses a set of loosely coupled black-box components, so called services. These services are orchestrated in order to offer some functionality to the user. A contract that comes with every service describes its functionality as well as the interface to be invoked. All these attributes of SOA enable the developer to design highly flexible distributed systems and re-use the components easily. In our approach for the design of DAS for truck-trailer-combinations, all functionalities offered by sensors, actors or assistance logic are encapsulated in services. These services are then re-orchestrated automatically whenever changes within the system occur. The services including their interfaces and contracts are specified in the UML profile SoaML using a process model based on "Service-oriented modeling and architecture" (SOMA, [2]). In [3], we refined SOMA in order to allow the development of embedded, automotive systems and added Quality of Service (QoS) parameters to the SoaML model.

Due to the automatic orchestration of the services the newly composed assistance system needs to be verified at runtime [4]. A DAS is a real-time system (RTS), its correctness depends (besides the functional correctness) mainly on the temporal properties of the components involved. Temporal Consistency [5] has been proposed as a quality criterion for real-time systems. If a system is temporal consistent wrt. a time period Δt, its outputs do not rely on any inputs older than Δt. Functionally correct assistance information that is based on old sensor data is potentially unsafe. To model and verify real-time properties, Hybrid Automata (HA) have proven to be useful in the past [6]. They are particularly well suited for Service Oriented real-time systems as they allow the modelling of system components individually and later compose them to an overall system automaton. HA provide a formal semantics and are thus accessible to formal methods – namely model checking (MC). There are two aspects of MC that outmatch other verification methods: the complete automation and the generation of an explanation in form of a counterexample in case of a failed verification attempt. In this paper, we present the approach to translate SoaML models annotated with QoS parameters to hybrid automata. Then, these automata can be checked for functional and real-time properties – like temporal consistency. Both steps can be executed completely automatically and are therefore predestined to be used at runtime to check newly orchestrated service compositions.

II. Hybrid Automata and Model Checking

A *Hybrid system* is a dynamical system that features both continuous and discrete state transitions. *Hybrid Automata* (HA, [7]) are a formal modelling language for hybrid systems. Informally, a hybrid automaton resembles a finite state machine. The state of a HA however does not solely consist of its current node but also of the valuation of a set of continuous variables. There are two kinds of state transitions: continuous transitions corresponding to the passage of time and discrete transitions corresponding to instant control node changes. A HA is augmented with mathematical expressions – invariant, flow, and jump constraints – to limit these state changes. The *invariant constraints* specify for each node, which valuations are admissible. The *flow constraints* limit the continuous state changes. They define for each node how the valuation can evolve over time and are given as a system of differential equations or inequalities. The *jump constraints* limit the discrete state changes. They determine for each edge

if it is currently enabled and how it changes the next valuation. Formally, the syntax is defined as follows:

Definition 1. A *hybrid automaton* is a 9-tuple $H = (Q,\ X, S, E, Inv, Flow, Jump, q_0, Init)$, where Q (the set of nodes), X (the ordered set of variables), S (the set of synchronisation labels) are finite and mutually disjoint and $E \subseteq Q \times Q \times S$ is the set of edges, Inv is a function assigning an invariant constraint over X to each $q \in Q$, $Flow$ is a function assigning a flow predicate over $\dot{X} \cup X$ to each $q \in Q$, $Jump$ is a function assigning a jump predicate over $X \cup X'$ to each $e \in E$, $q_0 \in Q$ is the initial node, and $Init$ is the initial predicate over X.

There exist many different classes of hybrid automata. For verification purposes, there has been much work on Linear Hybrid Automata (LHA, [7]). LHA restrict Inv, $Jump$, $Flow$, and $Init$ to conjunctions of linear inequalities , moreover $Flow$ may only contain variables from $\dot{X}$.

Given two hybrid automata, the composed automaton is derived purely syntactically. The composed automaton behaves like executing both automata concurrently. The components communicate on synchronization labels and shared variables.

The semantics of a hybrid automaton is defined in terms of a transition system between states. The *state* of a hybrid automaton is described by $(q, v) \in Q \times \mathbb{R}^{|X|}$ where v assigns a value to $x \in X$. The state is *admissible* iff $Inv(q)[v]$ holds. The state is *initial* iff $q = q_0$ and $Init[v]$ holds. There is a transition between two admissible states (q_1, v_1) and (q_2, v_2) if either there is a discrete transition (i.e. $\exists s \in S : e = (q_1, q_2, s) \in E \wedge Jump(e)[v_1, v_2]$) or there is a continuous transition (i.e. $q_1 = q_2$ and there is a function $f : \mathbb{R}_0^+ \to \mathbb{R}^n$ such that $Flow(q_1)[f]$ holds and $f(0) = v_1$ and $f(t) = v_2$ for some $t \in \mathbb{R}_0^+$) A state (q, v) is *reachable* if there is a sequence of states and transitions that starts in an initial state and contains (q, v).

Model checking [8] is a technique to fully automatically verify a system model (e.g. an automaton) versus a specification (given as a temporal logic formula). In the context of HA, model checking often refers to the verification of safety properties which is equivalent to reachability checking. If an unsafe state is reachable, the system is considered unsafe. The model checking algorithm performs a state exploration on the nodes and keeps track of the valuations symbolically. Due to the restriction to linear systems, all constraints can be translated to a linear inequality system (LIS). A symbolic state can be represented as a node together with a LIS. A discrete transition is enabled in a symbolic state, if the intersection of the inequality systems representing the state and the jump constraint is non-empty. This can be checked efficiently by linear programming. The model checking algorithm stops as soon as a fixed point is found or an unsafe state is reached. In the latter case, it outputs the sequence of states that led to the bad state, the so called *counterexample*.

III. Specifying SOA-based DAS

As described in section I, DAS for articulated vehicles are one of the scenarios where adaptive software and system architectures are needed. One of these DAS is a visual assistance for backing-up a truck with a one-axle trailer presented in [9]. The goal of this system is to add visual clues to the output of a rear view camera to help the driver estimating the vehicles movement. Therefore, the system measures the steering angle as well as the bending angle between truck and trailer. These values are then used to extrapolate the future trajectory of the trailer and of the overall truck-trailer-combination. In a next step, visual representations of these trajectories are overlayed on the picture of the rear view camera mounted on the back of the trailer and presented to the driver. In order to allow an automatic re-configuration of this system at runtime, it is built using the Service Oriented paradigm. Therefore the functionalities needed to offer the assistance are separated into services. We will use the following running example throughout the rest of this paper: The so called Overlay Service offers to overlay trajectories on a camera picture. It therefore invokes some other services to get a description of the trajectories as well as the current rear view picture. After the Overlay Service finished its task, it sends back the result to the component initially requested the information.

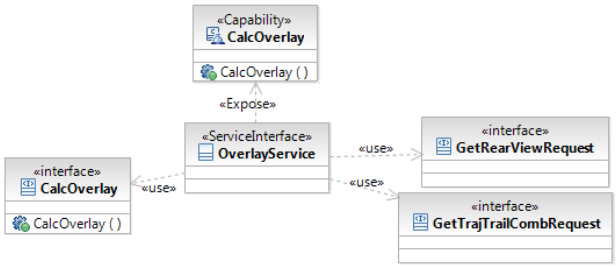

Fig. 1: The `ServiceInterface` of the Overlay Service.

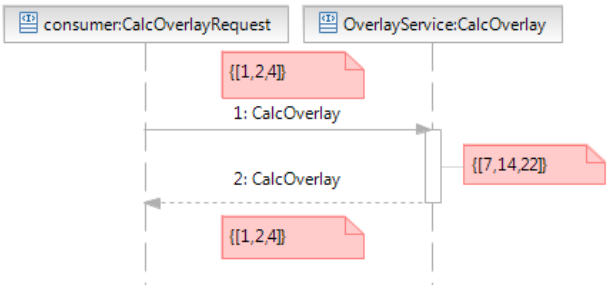

Fig. 2: The sequence chart showing the invocation of the Overlay Service along with the timing parameters.

In order to develop such services as well as Service Architectures a process model basing on SOMA has been set up in [3]. Starting from an informal description of the system it guides the developer creating a consistent description of the DAS in the Service Oriented modeling language (SoaML, [10]). SoaML is a Unified Modeling Language (UML) profile specified to model services as well as Service Oriented Architectures graphically. The profile aims on supporting the activities of service modelling and design. It therefore defines several SOA specific stereotypes. As the process model and SoaML itself are not in scope of this paper we focus on the elements of the description used for the verification. One of these elements is the `ServiceInterface`. It describes the functionalities offered by the service as well as the external functionalities needed to fulfil its task.

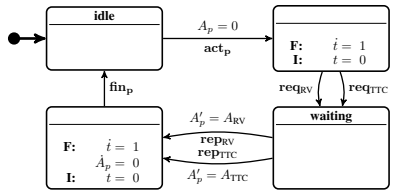

Fig. 3: Transformation of `ServiceInterface` of the Overlay Service (Fig. 1)

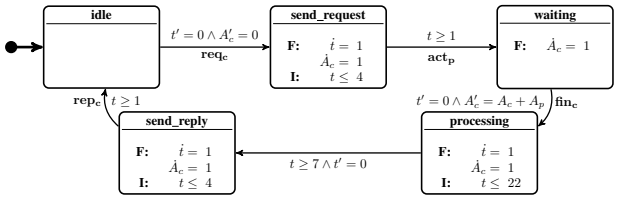

Fig. 4: Transformation of the `ServiceContract` of the Overlay Service (Fig. 2)

In Fig. 1 the `ServiceInterface` of the Overlay Service is illustrated. This service, which is derived from the necessity to have a functionality to calculate an overlay, offers one interface, namely 'CalcOverlay', which is pictured on the left side. The figure also shows that two other interfaces are needed to be invoked. These interfaces are to get the picture of the rear camera ('GetRearViewRequest') as well as the trajectories ('GetTrajTrailCombRequest'). Alongside with this information, a contract is specified describing the invocation process of every offered interface within a service. In order to do so, the SoaML stereotype `ServiceContract` defines the parties involved. In the case of the Overlay Service example, these are the interface offering the service ('CalcOverlay') as well as a consuming service ('CalcOverlayRequest'). A UML sequence diagram displayed in Fig. 2 models how the functionality is invoked.

In order to be able to carry out the qualitative verification we intent, Quality of Service (QoS) parameters have to be added to the model. For the sake of easy parsing and lucidity we are using constraints and mathematical notation to add these QoS parameters to the model. As we are focusing on the timing of the system, we decided to place the constraints directly into the sequence charts of the contracts. Figure 2 shows the sequence chart of the Overlay Service contract alongside with the constraints added. The format of the constraints added is [Minimum Time, Average Time, Maximum Time]. All times are given in milliseconds. As visible in Figure 2 every single element of the sequence is extended using a constraint. This kind of QoS notation is easily readable and parsable as well as precise and therefore particularly suitable in our context.

IV. Transformation and Verification

For the verification, the annotated SoaML models are transformed into hybrid automata. The transformation presented here is modular – each component of the SoaML model is translated independently. This allows a re-use of component automata and enhances traceability in case of a failed verification attempt.

We identified two classes of services for DAS namely event- and time-triggered. Event-triggered services are synchronous; a client sends a request and waits for a reply. This is the common service type for web services. For RTS this is not optimal due to additional delay. Time-triggered services send their output to a buffer periodically. They have a superior timely behavior but may spend resources like computation and network capacity unnecessarily. Being able to handle both service classes in one architecture allows to handle both – sporadic and periodical events. In order to treat both classes as uniform as possible, time-triggered services are handled as event-triggered ones by adding an additional component automaton sending requests periodically and serving as a buffer for the client.

Due to the lack of formal semantics of SoaML models, we have to make certain assumptions on the behavior of the specified services. We assume that when a service is executed, as a first step, it gathers all data needed to fulfill its task. Then there is some data processing step. Last, it sends its reply.

In the following, we describe how the artifacts of the SoaML model are transformed. The translation of a `ServiceInterface` is shown exemplarily in Fig. 3. This component automation stays in **idle** until it is activated externally by the synchronization label $\mathbf{act_p}$. Then it attempts to gather its inputs ($\mathbf{req_X}$), waits for an answer ($\mathbf{rep_X}$) and signals ($\mathbf{fin_p}$) that the data processing may begin. The sending and receiving is modelled using non-deterministic choice. This is valid because all possible choices are checked later and the system is safe only if all of them are safe. The translation of a `ServiceContract` is shown in Fig. 4. The contract automaton is **idle** until the service is requested ($\mathbf{req_c}$). It then simulates the time needed to send the request, signals (via $\mathbf{act_p}$) to start the gathering of inputs and waits for that to finish ($\mathbf{fin_p}$). Afterwards, the time needed to generate the output and to send it to the requester is simulated and it signals that the requesting interface automaton can continue ($\mathbf{rep_c}$). We decided to model the quality of data (in this case its age) explicitly as variables (A_x). This enables us to add more quality measures to the model in future work without changing the methodology.

For verification, the assistance designer has to provide safety conditions. In our example, each architecture contains a designated participant "assistance requester", which is annotated with a safety condition. It specifies the maximum age of data underlying the assistance output. Such a safety condition is related to the concept of temporal consistency [5] as explained in Section I.

V. Experimental Results

To validate our approach, we extended the tool *HieroMate* [11] to be able to import SoaML models and do the transformation as described in the previous section. *HieroMate* is a tool for graphical modelling of hybrid automata and serves as a front-end for the widely used model checker HyTech [12] and a model checking engine [13] based on Constraint Logic Programming (CLP).

We ran experiments containing the models of five different driver assistance systems containing 4 to 9 services each. The transformation resulted in 7 to 18 component automata with

up to 5 nodes each. Using the HyTech back-end, we were not able to prove safety. Given multiple component automata, HyTech computes the composition explicitly. This resulted in a blow-up of states and rendered verification impossible. The CLP implementation tries to avoid the state explosion at composition time by combining the component automata only as needed during the state exploration. As a result, the tool could prove safety for all of them in less than a second on a Intel Core i5-2557M CPU @ 2.70GHz with 4GB of RAM.

VI. Related Work

To the best of our knowledge, this is the first work that proposes the use of hybrid automata to model Service Oriented systems in the automotive domain. However, there are approaches to verify SOA-based software systems using model checking. One of these [14] forces the users to model their systems using a formal description language instead of a domain specific language. In contrast, another approach [15] translates activity diagrams from the UML4SOA profile to a formal language. A related approach [16] uses BPEL4WS to specify web service compositions, which are then translated to finite state processes. All these approaches share that they focus on web services and do not consider the special properties of real-time systems. Moreover, they are meant to be used at design time rather than checking the automatically orchestrated system at runtime. In contrast to these approaches, [17] proposes to validate the composed system at runtime. However, the approach favors using testing methods rather than formal verification which does not provide full coverage of system executions. Hybrid automata have recently been used to model Driver-Assistance-Systems [18]. Unfortunately, this approach considers a static configuration and focusses on the non-linear properties of their application. Timed automata have been used to verify UML sequence diagrams [19], [20]. However, `ServiceContracts` make only a very limited use of sequence diagrams and so a full translation of all features is not required in our context.

VII. Conclusion and Future work

With the workflow described above, we have shown that we are able to verify SOA-based Driver-Assistance-Systems. We also proved that our approach is capable of handling the two common execution scenarios, namely time-triggered and event-triggered. The results confirm that the whole transformation process from SoaML to hybrid automata as well as the model checking procedure can be executed in a timely manner.

In a next step the whole transformation and verification process could be ported to an embedded system that runs on the vehicle. This would allow to verify the newly orchestrated Assistance System at runtime. Besides that, the verification could be extended on other QoS parameters. One parameter to be tested could be the accuracy of the overall system which highly depends on the exactness of the services involved. In a last step, the results of the verification could be used to support the orchestration. This is because the technique of model checking not only outputs whether the verification process has been successful. In case of a failed verification, it also provides a counterexample containing a run of the hybrid automaton which caused the failure. Using this information, the services that should be replaced could be identified.

Acknowledgment

Marco Wagner has been supported by a grant of the Thomas Gessmann-Stiftung, Essen, Germany.

References

[1] M. Wagner, D. Zoebel, and A. Meroth, "Towards an adaptive software and system architecture for driver assistance systems," in *Computer Science and Information Technology (ICCSIT), 2011 4rd IEEE International Conference on*, 2011, pp. 174–178.

[2] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley, "Soma: A method for developing service-oriented solutions," *IBM Systems Journal*, vol. 47, no. 3, pp. 377 –396, 2008.

[3] M. Wagner, A. Meroth, and D. Zoebel, "Model-driven development of SOA-based driver assistance systems," in *Adaptive and Reconfigurable Embedded Systems, 1st Workshop (APRES 2012)*, 2012.

[4] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-Adaptive Software needs Quantitative Verification at Runtime," *Commun. ACM*, vol. 55, no. 9, pp. 69–77, 2012.

[5] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 2nd ed. Springer, 2011.

[6] R. Alur, T. Henzinger, and P. Ho, "Automatic symbolic verification of embedded systems," *Software Engineering, IEEE Transactions on*, vol. 22, no. 3, pp. 181–201, 1996.

[7] T. A. Henzinger, "The theory of hybrid automata," in *Proceedings of the 11th Annual Symposium on Logic in Computer Science*. IEEE Computer Society, 1996, pp. 278–292.

[8] E. Clarke, O. Grumberg, and D. Peled, *Model checking*. The MIT Press, 1999.

[9] U. Berg and D. Zoebel, "Visual steering assistance for backing-up vehicles with one-axle trailer," in *Vision in Vehicles 11*, 2006.

[10] *Service oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS) 1.0 Beta 2*, The Object Management Group, 2009.

[11] A. Mohammed and C. Schwarz, "HieroMate: A graphical tool for specification and verification of hierarchical hybrid automata," in *KI 2009, 32nd Annual German Conference on Artificial Intelligence*, ser. LNAI, vol. 5803. Springer, 2009, pp. 695–702.

[12] T. Henzinger, P.-H. Ho, and H. Wong-Toi, "HyTech: The Next Generation," in *IEEE Real-Time Systems Symposium*, 1995, pp. 56–65.

[13] A. Mohammed and U. Furbach, "Using CLP to model hybrid systems," in *Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming (CSCLP)*, 2008.

[14] I. Popovic, V. Vrtunski, and M. Popovic, "Formal verification of distributed transaction management in a soa based control system," in *Engineering of Computer-Based Systems, 18th IEEE International Conference and Workshops*. IEEE Computer Society, 2011, pp. 206–215.

[15] F. Banti, R. Pugliese, and F. Tiezzi, "An accessible verification environment for UML models of services," *Journal of Symbolic Computation*, vol. 46, no. 2, pp. 119–149, 2011.

[16] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-based verification of web service compositions," in *Automated Software Engineering, 18th IEEE International Conference*, 2003, pp. 152–161.

[17] T.-D. Cao, P. Felix, R. Castanet, and I. Berrada, "Online testing framework for web services," in *Software Testing, Verification and Validation, 3rd Int. Conference*. IEEE Computer Society, 2010, pp. 363–372.

[18] N. M. Enache, S. Mammar, M. Netto, and B. Lusetti, "Driver steering assistance for lane-departure avoidance based on hybrid automata and composite lyapunov function," *Trans. Intell. Transport. Sys.*, vol. 11, no. 1, pp. 28–39, 2010.

[19] T. Firley, M. Huhn, K. Diethers, T. Gehrke, and U. Goltz, "Timed sequence diagrams and tool-based analysis - a case study," in *UML'99: The Unified Modeling Language - Beyond the Standard, 2nd International Conference*, ser. LNCS, vol. 1723. Springer, 1999, pp. 645–660.

[20] M. Hendriks and M. Verhoef, "Timed automata based analysis of embedded system architectures," in *20th International Parallel and Distributed Processing Symposium*. IEEE, 2006.